

Ref #	Hits	Search Query	DBs	Default Operator	Plurals	Time Stamp
L1	26	717/112.CCLS.	US-PGPUB; USPAT; USOCR	OR	OFF	2005/07/14 15:53
S1	4	(state adj machine) and ((if near5 else) or if-else or if/else) and ((assembly adj language) or assembler or (machine adj code) )	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	OFF	2004/11/22 07:38
S2	51	(state adj machine) and (structured adj program\$4 )	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	OFF	2004/07/08 08:24
S3	23	(state adj machine) and (structured adj program\$4 ) and (assembly or assembler)	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	OFF	2004/07/08 08:34
S4	6	"5794040".URPN.	USPAT	OR	OFF	2004/07/08 08:28
S5	2724	assembly near3 (programming or language) and (if or if-else or if/else or (control adj flow) or constructs or conditional )	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2005/07/14 10:39
S6	359	assembly near3 (programming or language) and (if or if-else or if/else or (control adj flow) or constructs or conditional ) and (state adj machine)	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2004/07/08 08:37
S7	44	assembly near3 (programming or language) and (if or if-else or if/else or (control adj flow) or constructs or conditional ) same (state adj machine)	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2004/07/08 08:37
S8	359	(assembly near3 (programming or language) ) and (if or if-else or if/else or (control adj flow) or constructs or conditional ) and (state adj machine)	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2004/07/08 08:37
S9	44	(assembly near3 (programming or language) ) and (if or if-else or if/else or (control adj flow) or constructs or conditional ) same (state adj machine)	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2004/07/08 08:38

S10	40	((assembly near3 (programming or language) ) and (if or if-else or if/else or (control adj flow) or constructs or conditional ) same (state adj machine) ) not ((state adj machine) and (structured adj program\$4 ) and (assembly or assembler) )	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2004/07/08 09:19
S11	0	"6666383".URPN.	USPAT	OR	OFF	2004/07/08 08:44
S12	16	("5551033"   "5553232"   "5557737"   "5568380"   "5642516"   "5737586"   "5974525"   "5987258"   "6044449"   "6112273"   "6138230"   "6199095"   "6209082"   "6212609"   "6216200"   "6370640").PN.	USPAT	OR	OFF	2004/07/08 08:45
S13	0	"6718533".URPN.	USPAT	OR	OFF	2004/07/08 08:47
S14	20	("4484294"   "4568866"   "4796179"   "5055755"   "5086385"   "5404288"   "5546301"   "5838563"   "5971581"   "6026352"   "6076952"   "6106569"   "6119125"   "6134706"   "6154680"   "6195591"   "6226792"   "6259958"   "6425119"   "6477439").PN.	USPAT	OR	OFF	2004/07/08 08:47
S15	10	("5313614"   "5347654"   "5469572"   "5493675"   "5613117"   "5758061"   "5999737"   "6260190"   "6301704"   "6549930"   "2002/0032718"   "2002/0037496").PN.	USPAT	OR	OFF	2004/07/08 08:52
S16	6	"5347654".URPN.	USPAT	OR	OFF	2004/07/08 08:54
S17	6	("4598400"   "4773038"   "4827403"   "4984235"   "5129077"   "5187801").PN.	USPAT	OR	OFF	2004/07/08 08:54
S18	23	717/112.ccls.	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2004/07/08 09:21
S19	223	717/114.ccls.	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2004/07/08 09:21

S20	230	717/141.ccls.	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2004/07/08 09:22
S21	60	717/147.ccls.	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2004/07/08 13:51
S22	11	("4099230" "6179490" "6666383" "6718533" "6122356" ).pn.	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2004/07/08 13:51
S23	19952	assembler or (assembly adj language) or (assembly adj (code or program\$4))	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	OFF	2004/11/22 07:39
S24	18134	assembler or (assembly adj language) or (assembly adj (code or program\$4)) and (event or state) near5 (conditional or boolean)	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	OFF	2004/11/22 07:40
S25	18130	assembler or (assembly adj language) or (assembly adj (code or program\$4)) and (event or state) near5 (conditional or boolean) and (if or end or else or elseif)	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	OFF	2004/11/22 07:41
S26	18122	assembler or (assembly adj language) or (assembly adj (code or program\$4)) and (event or state) near5 (conditional or boolean) and (if or end or else or elseif) and (state adj machine)	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	OFF	2004/11/22 07:42
S27	17	(assembler or (assembly adj language) ) same ((event or state) near5 (conditional or boolean))	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	OFF	2004/11/22 07:43
S28	17	(assembler or (assembly adj language) ) same ((event or state) near5 (node or edge or graph) )	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2004/11/22 07:49
S29	665	(assembler or (assembly adj language) )and ((event or state) near5 (node or edge or graph) )	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2004/11/22 07:49

S30	28	(assembler or (assembly adj language) )and ((event or state) near5 (node or edge or graph) ) and ( 717/13? or 717/14? )	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2004/11/22 08:26
S31	27	S30 not S28	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2004/11/22 07:56
S32	4	("4885684" "5598564").pn.	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2004/11/22 07:56
S33	523	(assembler or (assembly adj language) ) and (control) and ( 717/13? or 717/14? )	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2004/11/22 08:26
S34	326	(assembler or (assembly adj language) ) and (control) and state and ( 717/13? or 717/14? )	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2004/11/22 08:35
S35	33	(assembler or (assembly adj language) ) and (control near3 state) and ( 717/13? or 717/14? )	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2004/11/22 08:36
S36	17	(assembly adj language) and (control near3 state) and ( 717/13? or 717/14? )	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2004/11/22 08:37
S37	1	"conditional program segment"	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	OFF	2005/07/13 15:45
S38	59	"conditional construct"	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	OFF	2005/07/13 15:45
S39	18	"conditional construct" and ( : setup or "set up")	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	OFF	2005/07/13 15:49

S40	0	"conditional construct" and (if near3 state)	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	OFF	2005/07/13 15:49
S41	0	"conditional construct" and (if near3 condition\$2)	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	OFF	2005/07/13 15:50
S42	0	"if state"	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	OFF	2005/07/13 15:50
S43	2	(if.near3 else ) same condition\$3 and construct and control\$4 and state	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2005/07/13 15:55
S44	40	("5579520" "5247693" "5991540" "5999735" "6023583" "6029005" "6031994" "6151704" "5276854" "5361351" "5511198" "5517645" "5581760" "5682536" "5689703" "5692122" "5699518" "5740439" "5742828" "5790778").pn.	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2005/07/13 16:10
S45	53	("5805885" "5842018" "6002879" "6016398" "6072950" "6106571" "6110227" "6202202" "6240465" "6243764" "6249907" "6263379" "6378126" "6412020" "646.470" "6466984" "4802116" "5864479" "4575816" "6421821" "5347654" "4809170" "4914659" "5671416" "5784553" "5611043" "57614708" "6314559" "6412106" "5991538").pn.	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2005/07/13 16:28
S46	8	( (structured adj (assembl\$2) ) adj (language or constructs or cod\$3 or program\$4) )	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2005/07/13 16:38
S47	6	("5774726" "6012836" "5598560").pn.	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2005/07/13 16:39

S48	16	("4667290"   "5179734"   "5269021"   "5280617"   "5339419"   "5432937"   "5488573"   "5572666"   "5581762").PN. OR ("5774726"). URPN.	US-PGPUB; USPAT; USOCR	OR	OFF	2005/07/14 10:37
S49	8530	assembly and macro	US-PGPUB; USPAT; USOCR	OR	ON	2005/07/14 10:38
S50	7	"synthetic instruction"	US-PGPUB; USPAT; USOCR	OR	ON	2005/07/14 10:38
S51	1154	assembly same macro	US-PGPUB; USPAT; USOCR	OR	ON	2005/07/14 10:38
S52	3366	assembly near3 (programming or language) and (if or if-else or if/else or (control adj flow) or constructs or conditional )	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2005/07/14 10:39
S53	142	S51 and S52	US-PGPUB; USPAT; EPO; JPO; DERWENT; IBM_TDB	OR	ON	2005/07/14 10:40
S54	2	("5598564").URPN.	USPAT	OR	OFF	2005/07/14 10:50
S55	15	("4315315"   "4587633"   "4620289"   "4750137"   "4805135"   "4831524"   "4928234"   "4942390"   "5129061"   "5185853"   "5191430"   "5195174"   "5207517"   "5208696"   "5222200").PN.	US-PGPUB; USPAT; USOCR	OR	OFF	2005/07/14 10:50
S56	5	("4553205"   "5640568"   "5999733").PN. OR ("6263493"). URPN.	US-PGPUB; USPAT; USOCR	OR	OFF	2005/07/14 11:31
S57	234	(setup or "set up") near5 macro	US-PGPUB; USPAT; USOCR	OR	OFF	2005/07/14 11:31
S58	44	(setup or "set up") near5 macro and assembly	US-PGPUB; USPAT; USOCR	OR	OFF	2005/07/14 11:32
S59	17	(setup or "set up") near5 macro and ((assembly or assembler) adj (language or program\$4 or code))	US-PGPUB; USPAT; USOCR	OR	OFF	2005/07/14 11:35

S60	130	(setup or "set up") near5 (macro or construct or directive or condition\$2 ) and ((assembly or assembler) adj (language or program\$4 or code))	US-PGPUB; USPAT; USOCR	OR	OFF	2005/07/14 11:36
S61	107	(setup or "set up") near3 (macro or construct or directive or condition\$2 ) and ((assembly or assembler) adj (language or program\$4 or code))	US-PGPUB; USPAT; USOCR	OR	OFF	2005/07/14 11:56
S62	2	meta-assembly	US-PGPUB; USPAT; USOCR	OR	OFF	2005/07/14 12:14
S63	1	"6061514".pn.	US-PGPUB; USPAT; USOCR	OR	OFF	2005/07/14 12:14
S64	6	("4831525"   "4860204"   "5212634"   "5761508"   "5790863"   "5854929").PN. OR ("6061514").URPN.	US-PGPUB; USPAT; USOCR	OR	OFF	2005/07/14 12:14
S65	6	("4831525"   "4860204"   "5212634"   "5761508"   "5790863"   "5854929").PN. OR ("6061514").URPN.	US-PGPUB; USPAT; USOCR	OR	OFF	2005/07/14 15:53




[Subscribe \(Full Service\)](#) [Register \(Limited Service, Free\)](#) [Log out](#)

 Search: ☒ The ACM Digital Library ☐ The Guide



THE ACM DIGITAL LIBRARY


[Feedback](#) [Report a problem](#) [Satisfaction survey](#)

 Terms used [if else macros](#)

Found 18,550 of 157

 Sort results  
by


[Save results to a Binder](#)
[Try an Advanced Search](#)

[Search Tips](#)
[Try this search in The ACM Guide](#)

 Display  
results

☐ Open results in a new  
window

Results 1 - 20 of 200

 Result page: [1](#) [2](#) [3](#) [4](#) [5](#) [6](#) [7](#) [8](#) [9](#) [10](#) [next](#)

Best 200 shown

 Relevance scale ☐ ☐ ☐

# 1 [The ML/I macro processor](#)

P. J. Brown

 October 1967 **Communications of the ACM**, Volume 10 Issue 10

 Full text available: [pdf\(844.57 KB\)](#) Additional Information: [full citation](#), [references](#), [citations](#), [index terms](#)

# 2 [Macros that work](#)

William Clinger

 January 1991 **Proceedings of the 18th ACM SIGPLAN-SIGACT symposium on Principles of programming languages**

 Full text available: [pdf\(802.33 KB\)](#) Additional Information: [full citation](#), [references](#), [citations](#), [index terms](#)

# 3 [Session 1 \(full technical papers\): evolution in source code: Challenges of refactoring C programs](#)

Alejandra Garrido, Ralph Johnson

 May 2002 **Proceedings of the International Workshop on Principles of Software Evolution**

 Full text available: [pdf\(687.83 KB\)](#) Additional Information: [full citation](#), [abstract](#), [references](#), [index terms](#)

Refactoring has become a well-known technique for transforming code in a way that preserves behavior. Refactorings may be applied manually, although manual code manipulation is error prone and cumbersome, so maintainers need tools to make automatic refactorings. There is currently extensive literature on refactoring object-oriented programs and some very good tools for refactoring Smalltalk and Java code. Although there is more code written in C or C++ than in any other language, refactoring too ...

**Keywords:** C programming, preprocessor directives, refactoring


# 4

[Module generation of complex macros for logic-emulation applications](#)



Wen-Jong Fang, Allen C.-H. Wu, Duan-Ping Chen


February 1997 **Proceedings of the 1997 ACM fifth international symposium on Field-programmable gate arrays**

Full text available:  [pdf\(1.48 MB\)](#) Additional Information: [full citation](#), [references](#), [citations](#), [index terms](#)

5 The scheme of things: implementing lexically scoped macros

Jonathan Rees

January 1993 **ACM SIGPLAN Lisp Pointers**, Volume VI Issue 1


Full text available:  [pdf\(257.50 KB\)](#) Additional Information: [full citation](#), [abstract](#), [index terms](#)

I have been hearing some complaints that Scheme's new lexically scoped macro facility is difficult to implement. There are two components to the proposal: the pattern language and lexical scoping. The two pose independent problems. I agree that the ellipsis-enriched pattern language can be tricky to implement; implementations that I have seen take anywhere from 250 to 1400 lines of Scheme code. However, I believe that it is conceptually straightforward, and several implementations have been around ...

6 Typesetting with groff Macros

Wayne Marshall

November 2000 **Linux Journal**


Full text available:  [html\(27.61 KB\)](#) Additional Information: [full citation](#), [abstract](#), [references](#), [index terms](#)

Reports of troff's death are greatly exaggerated.

7 Macros as multi-stage computations: type-safe, generative, binding macros in MacroML

Steven E. Ganz, Amr Sabry, Walid Taha

October 2001 **ACM SIGPLAN Notices , Proceedings of the sixth ACM SIGPLAN international conference on Functional programming**, Volume 36 Issue 10


Full text available:  [pdf\(233.27 KB\)](#) Additional Information: [full citation](#), [abstract](#), [references](#), [citations](#), [index terms](#)

With few exceptions, macros have traditionally been viewed as operations on syntax trees or even on plain strings. This view makes macros seem ad hoc, and is at odds with two desirable features of contemporary typed functional languages: static typing and static scoping. At a deeper level, there is a need for a simple, usable semantics for macros. This paper argues that these problems can be addressed by formally viewing macros as multi-stage computations. This view eliminates the need for fresh ...

8 A Proposal for Macro-Facilities in ALGOL

H. Leroy


February 1966 **ALGOL Bulletin**, Issue 22

Full text available:  [pdf\(629.94 KB\)](#) Additional Information: [full citation](#), [index terms](#)

9 Structured Assembly language in VAX-11 MACRO

Robert R. Leeper, Karl O. Rehmer

February 1986 **ACM SIGCSE Bulletin , Proceedings of the seventeenth SIGCSE technical symposium on Computer science education, Volume 18 Issue 1**


Full text available:  [pdf\(499.69 KB\)](#) Additional Information: [full citation](#), [abstract](#), [references](#), [index terms](#)

For several years, the introductory assembly language course at Indiana-Purdue at Fort Wayne has used "structured" assembly language on an IBM System 370. A later course makes use of VAX-11 MACRO assembly language on a VAX 11/780. This paper shows how the major constructs for structured programming may be implemented in VAX-11 MACRO. The scheme involves assembly language templates for each of the constructs, a standard labeling scheme, and a commenting method which reflects the ...

**10 Dissolve transition detection algorithm using spatio-temporal distribution of MPEG macro-block types (poster session)**

Sung-Bae Jun, Kyoungro Yoon, Hee-Youn Lee

October 2000 **Proceedings of the eighth ACM international conference on Multimedia**

Full text available:  [pdf\(394.68 KB\)](#) Additional Information: [full citation](#), [abstract](#), [references](#), [index terms](#)


Almost every shot change detection algorithm detects abrupt transition (hard cut) without difficulty, but gradual transitions such as fades, dissolves, wipes are left as hard-to-detect problems. Dissolve effect, among the various gradual transition effects, is one of the most frequently used shot transition methods with special semantic meaning such as scene transition. Information of the shot change type can also be the basis for the shot clustering algorithms. In this paper, we present a fa ...

**Keywords:** MPEG, dissolve, fades, macro block type distribution, shot change detection, video segmentation

**11 Algorithm 803: a simpler macro processor**

William A. Ward

June 2000 **ACM Transactions on Mathematical Software (TOMS), Volume 26 Issue 2**

Full text available:  [pdf\(73.41 KB\)](#) Additional Information: [full citation](#), [abstract](#), [references](#), [index terms](#)


Macro processors have been in the computing tool chest since the late 1950's. Their use, though perhaps not what it was in the heyday of assembly language programming, is still widespread. In the past, producing a full-featured macro processor has required significant effort, similar to that required to implement the front-end to a compiler augmented by appropriate text substitution capabilities. The tool described here adopts a different approach. The text containing macro definitions and ...

**Keywords:** awk, portable, simple

**12 BIGMAC II: A FORTRAN language augmentation tool**

Eugene W. Myers, Leon J. Osterweil

March 1981 **Proceedings of the 5th international conference on Software engineering**

Full text available:  [pdf\(1.00 KB\)](#) Additional Information: [full citation](#), [abstract](#), [references](#), [citations](#)

[MB\)](#)[index terms](#)

This paper describes the motivation, design, implementation, and some preliminary performance characteristics of BIGMAC II, a macro definition capability for creating language enhancers and translators. BIGMAC II enables the user to specify transformations through STREX, a FORTRAN-like language, which enables the specification of macros which are then used to interpretively alter incoming programs. BIGMAC II is specially adapted to the processing of FORTRAN programs. This paper shows how it ...

### 13 [A survey of the systematic use of macros in systems building](#)

David J. Farber

October 1971 **ACM SIGPLAN Notices , Proceedings of the SIGPLAN symposium on Languages for system implementation**, Volume 6 Issue 9

Full text available: [pdf\(494.89 KB\)](#) Additional Information: [full citation](#), [abstract](#), [references](#), [index terms](#)

Assemblers with macro capabilities have been available for over ten years. There has been a limited number of mainly unpublished systematic uses of such capabilities in the construction of a variety of systems. This paper will cover a number of these cases. We will examine the features of the macro systems which allowed their usage as well as the method used in the system implementation. We will comment on the effect on efficiency and flexibility that the use of this facility has produced.< ...

### 14 [Microinstruction sequencing and structured microprogramming](#)

Louise H. Jones

September 1974 **Conference record of the 7th annual workshop on Microprogramming**

Full text available: [pdf\(651.54 KB\)](#) Additional Information: [full citation](#), [abstract](#), [references](#), [citations](#), [index terms](#)

The purpose of this paper is to explore the relation between the sequencing functions of microprogrammable computers and the implementation of the control constructs of flowchartable logic with modular microcode. So far, there has been little or no discussion of this topic in the literature [8]. Sequencing functions for various microprogrammable processors are described in Section 2; the implementation of the Mills' control constructs using various sets of sequencing functions is discussed i ...

### 15 [The remaining trouble spots in ALGOL 60](#)

Donald E. Knuth

October 1967 **Communications of the ACM**, Volume 10 Issue 10

Full text available: [pdf\(1.37 MB\)](#) Additional Information: [full citation](#), [references](#), [citations](#), [index terms](#)

### 16 [Concise reference manual for the Series macro package](#)

Richard C. Waters

July 1989 **ACM SIGPLAN Lisp Pointers**, Volume III Issue 1

Full text available: [pdf\(1.85 MB\)](#) Additional Information: [full citation](#), [abstract](#), [index terms](#)


Series expressions are transformed into loops by pipelining them---the computation is converted from a form where entire series are computed one after the other to a form where the series are

incrementally computed in parallel. In the resulting loop, each individual element is computed just once, used, and then discarded before the next element is computed. For this pipelining to be possible, four restrictions have to be satisfied. Before looking at these restrictions, it is useful to consider a ...

**17** Model checking XML manipulating software

Xiang Fu, Tevfik Bultan, Jianwen Su

July 2004 **ACM SIGSOFT Software Engineering Notes , Proceedings of the 2004 ACM SIGSOFT international symposium on Software testing and analysis, Volume 29 Issue 4**

Full text available:  [pdf\(199.58 KB\)](#) Additional Information: [full citation](#), [abstract](#), [references](#), [index terms](#)


The use of XML as the de facto data exchange standard has allowed integration of heterogeneous web based software systems regardless of implementation platforms and programming languages. On the other hand, the rich tree-structured data representation, and the expressive XML query languages (such as XPath) make formal specification and verification of software systems that manipulate XML data a challenge. In this paper, we present our initial efforts in automated verification of XML data manipul ...

**Keywords:** MSL, SPIN, XML, XML schema, XPath, model checking, promela, web service

**18** Compact list representation: definition, garbage collection, and system implementation

Wilfred J. Hansen

September 1969 **Communications of the ACM, Volume 12 Issue 9**

Full text available:  [pdf\(1.19 MB\)](#) Additional Information: [full citation](#), [abstract](#), [references](#), [citations](#), [index terms](#)

Compact lists are stored sequentially in memory, rather than chained with pointers. Since this is not always convenient, the Swym system permits a list to be chained, compact, or any combination of the two. A description is given of that list representation and the operators implemented (most are similar to those of LISP 1.5). The system garbage collector attempts to make all lists compact; it relocates and rearranges all of list storage using temporary storage. This unique list-compacting ...

**Keywords:** LISP, compact list, data representation, data structure, free storage, garbage collection, list, list processing system, list representation, list structure, macro, plex, plex processing, pointer, primitive list operations, relocation, storage reclamation

**19** An improved hands-on approach to teaching systems programming and the impact of structured programming

Roger T. Cooper, Malcolm G. Lane

July 1976 **ACM SIGCSE Bulletin , Proceedings of the sixth SIGCSE technical symposium on Computer science education, Volume 8 Issue 3**

Full text available:  [pdf\(703.40 KB\)](#) Additional Information: [full citation](#), [abstract](#), [references](#), [citations](#), [index terms](#)


The use of the hands-on approach for teaching systems programming presented at the 1974 SIGCSE Conference has proved to be even more successful in the past two years. The reasons for

the increased success are given. An approach of using structured assembler language concepts as an integral part of the systems programming course is introduced and discussed. Specific examples of the use of several structured programming macros are presented.

**20 SLANG a problem solving language for continuous-model simulation and optimization**

Joe M. Thames

August 1969 **Proceedings of the 1969 24th national conference**

Full text available:  pdf(1.00 MB) Additional Information: [full citation](#), [abstract](#), [references](#), [citations](#), [index terms](#)





SLANG is a mathematical problem modeling and solution language. It is one of several languages in the programming subsystem of the Computer User Executive (CUE) System developed by TRW Systems. SLANG is both a procedural and a command language designed primarily for the "casual" user. Consequently, much attention was paid to programming ease, "natural" syntax rules, readability, and debugging ease. On the other hand, SLANG is designed to permit the solution of very s ...

Results 1 - 20 of 200

Result page: [1](#) [2](#) [3](#) [4](#) [5](#) [6](#) [7](#) [8](#) [9](#) [10](#) [next](#)

The ACM Portal is published by the Association for Computing Machinery. Copyright © 2005 ACM Inc.

[Terms of Usage](#) [Privacy Policy](#) [Code of Ethics](#) [Contact Us](#)

Useful downloads:  [Adobe Acrobat](#)  [QuickTime](#)  [Windows Media Player](#)  [Real Playe](#)


[Subscribe \(Full Service\)](#) [Register \(Limited Service, Free\)](#) [Log out](#)

 Search: ☒ The ACM Digital Library ☐ The Guide

THE ACM DIGITAL LIBRARY

[Feedback](#) [Report a problem](#) [Satisfaction survey](#)

 Terms used **extending assembly macro**

 Found **10,340** of **157**

 Sort results by 
☒ [Save results to a Binder](#)
[Try an Advanced Search](#)
☒ [Search Tips](#)
[Try this search in The ACM Guide](#)

 Display results 
☐ [Open results in a new window](#)

Results 1 - 20 of 200

 Result page: [1](#) [2](#) [3](#) [4](#) [5](#) [6](#) [7](#) [8](#) [9](#) [10](#) [next](#)

Best 200 shown

 Relevance scale ☐ ☒ ☐

### 1 [Translator writing systems](#)

Jerome Feldman, David Gries

 February 1968 **Communications of the ACM**, Volume 11 Issue 2

 Full text available: ☒ pdf(4.47 MB)

 Additional Information: [full citation](#), [abstract](#), [references](#), [citations](#)

A critical review of recent efforts to automate the writing of translators of programming languages is presented. The formal study of syntax and its application to translator writing are discussed in Section II. Various approaches to automating the postsyntactic (semantic) aspects of translator writing are discussed in Section III, and several related topics in Section IV.

**Keywords:** compiler compiler-compiler, generator, macroprocessor, meta-assembler, metacompiler, parser, semantics, syntactic analysis, syntax, syntax-directed, translator, translator writing system

### 2 [Structured programming in assembly language](#)

Andries van Dam, Jens M. Dill, Douglas F. Dixon, David S. Notkin

 December 1976 **ACM SIGCSE Bulletin**, Volume 8 Issue 4

 Full text available: ☒ pdf(871.34 KB)

 Additional Information: [full citation](#), [abstract](#), [citations](#), [index terms](#)

Structured design and programming techniques can be extended from high-level languages to assembly language. Over the past three years at Brown University, beginning assembly language programmers have been successfully taught these techniques using clearly defined standards. These standards and the solutions to several of the typical problems that arise in structured assembly language programming are discussed in this paper.

### 3 [Macro processing in high-level languages](#)

Alexander Sakharov

 November 1992 **ACM SIGPLAN Notices**, Volume 27 Issue 11

 Full text available: ☒ pdf(709.71 KB)


 Additional Information: [full citation](#), [abstract](#), [index terms](#)

A macro language is proposed. It enables macro processing in high-level programming languages. Macro definitions in this language refer to the grammars of the respective programming languages. These macros introduce new constructs in programming languages. It is described how to automatically generate macro processors from macro definitions and programming language grammars written in the lex-yacc format. Examples of extending high-level languages by means of macros are given.

4 A use of macros in translation of symbolic assembly language of one computer to another

George T. Dellert


December 1965 **Communications of the ACM**, Volume 8 Issue 12

Full text available:  [pdf\(874.72 KB\)](#) Additional Information: [full citation](#), [references](#), [citations](#), [index terms](#)

5 From system F to typed assembly language

Greg Morrisett, David Walker, Karl Crary, Neal Glew

May 1999 **ACM Transactions on Programming Languages and Systems (TOPLAS)**, Volume 21 Issue 3

Full text available:  [pdf\(483.91 KB\)](#) Additional Information: [full citation](#), [abstract](#), [references](#), [citations](#), [index terms](#)


We motivate the design of typed assembly language (TAL) and present a type-preserving translation from System F to TAL. The typed assembly language we present is based on a conventional RISC assembly language, but its static type system provides support for enforcing high-level language abstractions, such as closures, tuples, and user-defined abstract data types. The type system ensures that well-typed programs cannot violate these abstractions. In addition, the typing constructs admit ...

**Keywords:** certified code, closure conversion, secure extensible systems, type-directed compilation, typed assembly language, typed intermediate languages

6 A graded bibliography on macro systems and extensible languages

John R. Metzner

January 1979 **ACM SIGPLAN Notices**, Volume 14 Issue 1

Full text available:  [pdf\(830.04 KB\)](#) Additional Information: [full citation](#), [references](#)

7 Conference abstracts

January 1977 **Proceedings of the 5th annual ACM computer science conference**

Full text available:  [pdf\(3.14 MB\)](#) Additional Information: [full citation](#), [abstract](#), [index terms](#)

One problem in computer program testing arises when errors are found and corrected after a portion of the tests have run properly. How can it be shown that a fix to one area of the code does not adversely affect the execution of another area? What is needed is a quantitative method for assuring that new program modifications do not introduce new errors into the code. This model considers the retest philosophy that every program instruction that could possibly be reached and




tested from the ...

8 A history of the SNOBOL programming languages

Ralph E. Griswold

January 1978 **ACM SIGPLAN Notices**, **The first ACM SIGPLAN conference on History of programming languages**, Volume 13 Issue 8


Full text available:  [pdf\(3.56 MB\)](#) Additional Information: [full citation](#), [abstract](#), [references](#), [index terms](#)

Development of the SNOBOL language began in 1962. It was followed by SNOBOL2, SNOBOL3, and SNOBOL4. Except for SNOBOL2 and SNOBOL3 (which were closely related), the others differ substantially and hence are more properly considered separate languages than versions of one language. In this paper historical emphasis is placed on the original language, SNOBOL, although important aspects of the subsequent languages are covered.

9 A user-oriented macro processor

D. L. Marrs, C. W. Collins, H. A. Hess

January 1968 **Proceedings of the 1968 23rd ACM national conference**


Full text available:  [pdf\(932.92 KB\)](#) Additional Information: [full citation](#), [abstract](#), [references](#), [citations](#), [index terms](#)

Machine-language assemblers map statements in a source language to the machine language of a particular computer. The source language accepted by the assembler is, for the most part, defined in a one-to-one manner with the hardware instructions available on the target computer. The addition of a macro processor to the assembler, referred to then as a macro assembler, effectively extends the range of the source languages accepted by the assembler. This is achieved by providing the ...

10 Signal corps research and development on automatic programming of digital computers

William F. Luebbert, Percy W. Collom


February 1959 **Communications of the ACM**, Volume 2 Issue 2

Full text available:  [pdf\(751.66 KB\)](#) Additional Information: [full citation](#)

11 The ML/I macro processor

P. J. Brown


October 1967 **Communications of the ACM**, Volume 10 Issue 10

Full text available:  [pdf\(844.57 KB\)](#) Additional Information: [full citation](#), [references](#), [citations](#), [index terms](#)

12 Special issue on programming language design: Extensibility in programming language design

Thomas A. Standish


July 1975 **ACM SIGPLAN Notices**, Volume 10 Issue 7

Full text available:  [pdf\(368.13 KB\)](#) Additional Information: [full citation](#), [references](#)

13 A contingency approach to the application software generations

Roger Clarke

June 1991 **ACM SIGMIS Database**, Volume 22 Issue 3

Full text available:  [pdf\(1.26 MB\)](#)


Additional Information: [full citation](#), [abstract](#), [index terms](#)

The current environment in which software development is undertaken includes a mix of languages at varying levels of abstraction. The goal of this paper is threefold. First, the range of application software technologies is reviewed from a historical perspective. Second, a model based on levels of abstraction summarizes the key differences between these technologies. Third, a contingency model is proposed to guide the selection of the appropriate level of abstraction. It is argued that the selecti ...

14 Syntax translation with context macros or macros without arguments

Charles A. Grant

September 1971 **ACM SIGPLAN Notices , Proceedings of the international symposium on Extensible languages**, Volume 6 Issue 12

Full text available:  [pdf\(149.20 KB\)](#)


Additional Information: [full citation](#), [abstract](#), [references](#), [citations](#), [index terms](#)

A conceptually language independent macro facility with the following properties is described here: 1) Each macro gathers its own arguments from the text in which the call is embedded (the context). 2) A macro may modify its context an arbitrary distance from the actual string which represents the call. 3) A macro may specify the precise point from which scanning is to continue after the macro returns. The objective of this work is to define a simp ...

15 Technical contributions: BUILD: a primitive approach to the design of computer languages and their translators

Richard K. Bennett

July 1976 **ACM SIGPLAN Notices**, Volume 11 Issue 7

Full text available:  [pdf\(642.26 KB\)](#)

Additional Information: [full citation](#), [abstract](#), [references](#)


The traditional approach to computer language design has been to design one language at a time. This approach has led to languages which are syntactically complex and incompatible with one another. The proposed approach develops a language base, utilizing primitive language forms, upon which new languages may be built for all purposes. The resulting languages may possess diverse vocabulary and characteristics, but they are unified by the grammar of the language base upon which they are built, and ...

**Keywords:** Assemblers, Compilers, Computer Languages, Extensible Languages, Language Base, Language Definition, Languages, Programming Languages, Translators

16 Teaching assembly language: a comparison of IBM S/360 and Intel 80x86 courses

Richard C. Detmer


February 1990 **ACM SIGCSE Bulletin , Proceedings of the twenty-first SIGCSE technical symposium on Computer science education**, Volume 22 Issue 1

Full text available:  [pdf\(550.16 KB\)](#)

Additional Information: [full citation](#), [references](#), [index terms](#)

**17 A survey of the systematic use of macros in systems building**


David J. Farber

October 1971 **ACM SIGPLAN Notices , Proceedings of the SIGPLAN symposium on Languages for system implementation**, Volume 6 Issue 9Full text available:  [pdf\(494.89 KB\)](#) Additional Information: [full citation](#), [abstract](#), [references](#), [index terms](#)

Assemblers with macro capabilities have been available for over ten years. There has been a limited number of mainly unpublished systematic uses of such capabilities in the construction of a variety of systems. This paper will cover a number of these cases. We will examine the features of the macro systems which allowed their usage as well as the method used in the system implementation. We will comment on the effect on efficiency and flexibility that the use of this facility has produced.< ...

**18 Syntax macros and extended translation**


B. M. Leavenworth

November 1966 **Communications of the ACM**, Volume 9 Issue 11Full text available:  [pdf\(628.14 KB\)](#) Additional Information: [full citation](#), [abstract](#), [references](#), [citations](#)

A translation approach is described which allows one to extend the syntax and semantics of a given high-level base language by the use of a new formalism called a syntax-macro. Syntax-macros define string transformations based on syntactic elements of the base language. Two types of macros are discussed, and examples are given of their use. The conditional generation of macros based on options and alternatives recognized by the scan are also described.

**19 Approaches to design of high level languages for microprogramming**


Patrick W. Mallett, T. G. Lewis

September 1974 **Conference record of the 7th annual workshop on Microprogramming**Full text available:  [pdf\(605.30 KB\)](#) Additional Information: [full citation](#), [abstract](#), [references](#), [citations](#), [index terms](#)

The development of a programming language for microprocessors (we will use the term microprocessor to refer to the hardware host to be microprogrammed) differs from the development of languages for conventional processors [5, 6]. The difference is traced to: (1) concurrency of hardware (2) limited writable local memory (3) main memory access delay and I/O control (4) limited microprocessor instruction repertoire (5) special hardware cha ...

**20 A data-flow driven resource allocation in a retargetable microcode compiler**

H. Feuerhahn

January 1988 **Proceedings of the 21st annual workshop on Microprogramming and microarchitecture**Full text available:  [pdf\(328.53 KB\)](#) Additional Information: [full citation](#), [abstract](#), [references](#), [citations](#), [index terms](#)

A method for global resource allocation is described, which minimizes data movements and optimizes the use of resources like special purpose registers and functional units in complicated bus structures. The algorithm can deal with arbitrary flow graphs and hierarchies of nonrecursive procedures. It is based on a thorough data flow analysis of the source program and a description of the target architecture. The method has been implemented in a retargetable compiler with front-

ends ...

Results 1 - 20 of 200

Result page: [1](#) [2](#) [3](#) [4](#) [5](#) [6](#) [7](#) [8](#) [9](#) [10](#) [next](#)

The ACM Portal is published by the Association for Computing Machinery. Copyright © 2005 ACM Inc.

[Terms of Usage](#) [Privacy Policy](#) [Code of Ethics](#) [Contact Us](#)

Useful downloads:  [Adobe Acrobat](#)  [QuickTime](#)  [Windows Media Player](#)  [Real Playe](#)